commodore

# The Transactor

Commodore Canada's
Tech/News Periodical

VOLUME 3
Issue #3

## Bits & Pieces

### Unit to Unit Copying

Ever needed to copy one or two programs from 4040 to 8050 (or vice versa)? So you pull out your Change Unit Address program, then Copy All, and after answering all the prompts, the files you've selected get transfered. Seems like a lot of work for only a couple of files, doesn't it.

Alas, there is another way! Connect your 4040 and 8050 to the PET/CBM and leave the device numbers alone (ie. both device 8). Assuming we're going from 4040 to 8050, insert your source disk in drive 0 of the 4040 and place the destination disk in drive 1 of the 8050. Now DLOAD the program from 4040 drive 0. Both disk units will fire up, but the 8050 will give an error since drive 0 is empty... So What! DSAVE to drive 1 and the 8050 error light will go off and the 4040 error light comes on. Again... So What! Check the directory and you'll find the transfer took place without a hitch!

This would also work for SEQ files with a small bit of software that knows how to ignore anticipated errors. Of course, for a lot of files or programs, the Copy All approach would probably be quicker, but isn't it nice to know you can deliberately cause disk errors and still accomplish something!

### IEEE Timeout Defeat

IEEE Timeout is a condition that simply says, "this bus operation took too long!". There are two such situations; Timeout on Read and Timeout on Write.

Timeout on Read has occurred when ST=2. This usually happens when you try to read past the end of an SEQ file. More generally, the controller (PET) has asked a peripheral for a byte and the peripheral did not respond with that byte within the 64 ms. time limit.

The Transactor is produced on the CBM 8032 with WordPro IV Plus and the NEC Spinwriter

Index Transactor #3


Bits & Pieces ..... 1
  Unit to Unit Copying ..... 1
  IEEE Timeout Defeat ..... 1
  Comal-80 Is Here! ..... 3
  Pacific Coast Computer Fair ..... 4
  CBM 8010 Auto-Answerer ..... 4
Backup ..... 6
  Disable RUN/STOP ..... 6
  Joystick/Keyboard Routine ..... 6
Programs to WordPro Converter ..... 7
8032 / DOS 2.0 Tips ..... 8
Jim Strasma's SUPERSORT ..... 15
Jim Butterfield on Programming ..... 16
Screen Editting ..... 18
SWARM-100 For PET/CBM ..... 21
Bulletin Boards ..... 26
Calculator Revisited ..... 28
Spell Checking Programs ..... 34
Toronto PET Users Group Memberships .. 39

Timeout on Write (ST=1) will happen if you write to a file that is open in the PET but not open in the peripheral. For example, OPEN an SEQ disk file for writing and reset the disk or perform a CATALOG command. PET still considers this file open while the disk has gone and closed it. Try a PRINT# now and Timeout on Write will be flagged.

This protocal works great with Commodore peripherals, but unfortunately there exist devices with much slower response times. The biggest offenders are X-Y Plotters.

Fortunately, in all BASIC 4.0 PET/CBMs, there is a feature that allows you to disable IEEE Timeout. When the bus flags timeout, BASIC 4.0 checks location 1020 (hex 03fc) to determine how to handle it.

POKE 1020, 255

If 1020 contains a negative number (bit 7 set), PET will wait forever for the peripheral to accept the byte last delivered to the bus. Be careful though... the PET can hang up if your peripheral doesn't accept this character. This might sound somewhat precarious but it has its advantages. If your peripheral is receiving characters successfully, but just not fast enough, disable timeout and no data will be lost.

To enable timeouts POKE 1020, 0.

## Comal-80 Is Here!

Comal-80 is a programming language that supports very structured programming techniques. Comal works on CBM 8032s only. It loads into RAM and and virtually gives you a new operating system.

Comal has been approved as the learning programming language of Holland and already there is a Comal Users Group based in the U.S.

The best application for Comal is in the educational environment. Teachers will find that introducing programming via Comal is an excellent first step.

The best feature in Comal is the price... it's FREE! All authorized Commodore dealers in Canada have the package and the disk & documentation are available for copying. COMAL stands for COMmon Algorithmic Language.

The address for the Comal Users Group is:

Comal Users Group
5501 Groveland Terrace
Madison, Wisconson
53716

Send a large self-addressed envelope with 35 cents U.S. for return postage and receive additional Comal info. More on Comal in future issues of The Transactor.

## Pacific Coast Computer Fair

The Pacific Coast Computer Fair Association is presenting Vancouver's second annual Computer Fair. This years theme is "Computers and the Handicapped" in recognition of the International Year of the Disabled.

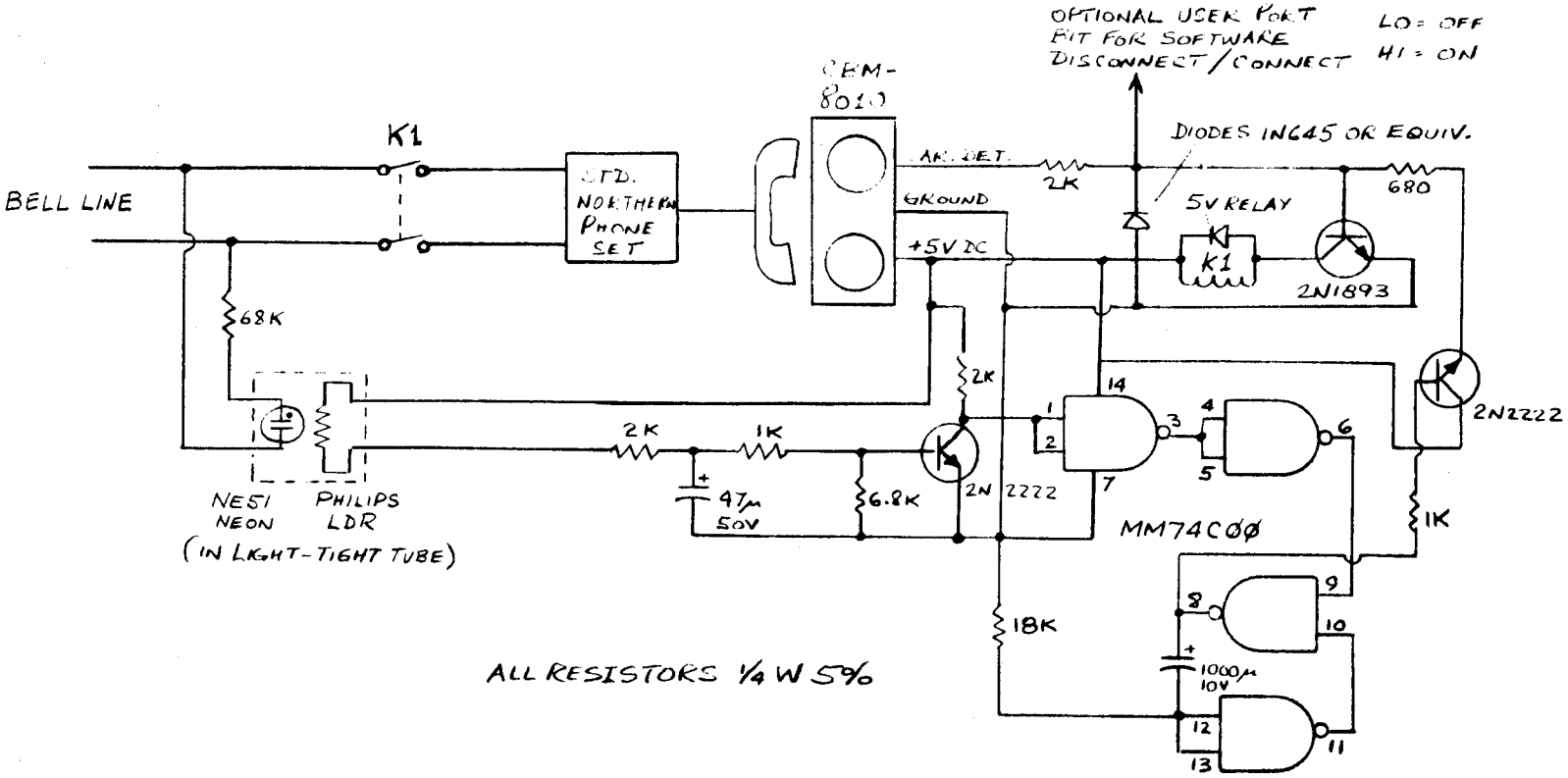
Site of the fair is The Robson Square Media Centre in downtown Vancouver. With 65,000 sq. ft. and over 50 exibitors, the PCCFA is expecting more than 10,000 visitors! Show dates are October 3rd & 4th.

A second conference is slated for Oct. 2nd. This will be closed to the public. For more information, contact:

PCCFA
4100 St, Goerges Ave.
North Vancouver, B.C.
V7N 1W8

## CBM 8010 Auto-Answer Circuit

The schematic on the following page is an automatic answering device for the CBM 8010 IEEE Modem. It was designed by Dieter Demmer of Oakville Ontario. The circuit hasn't been tested by us, but presumably it works for Dieter!




CAR. DET. = + LEAD FROM CRX LED.

# AUTOMATIC ANSWER FOR CBM 8010 MODEM

D. DEMMER (416) 844-4060

## OPERATION:

1. RING DETECTOR CONNECTS LINE AT FIRST RING.
2. MODEM TRANSMITS CARRIER (ANSWER-MODE).
3. IF ORIGINATE CARRIER IS RECEIVED, RELAY LATCHES.
4. IF NONE RECEIVED FOR 10 SECS., LINE DISCONNECTS AND IS ARMED FOR NEXT RINGING.

   NOTE: CBM MODEM MUST BE IN ANSWER MODE!
5. OPTIONAL SOFTWARE DISCONNECT SETS PORT BIT LOW.

## Disable RUN/STOP

In Transactor #2 of Volume 3, Jim Butterfield's disable STOP key routine was published; the one that does not disable the clock. The code is correct but you need one more POKE to engage it:

```
POKE 145, 3
```

The above POKE points the IRQ vector at Jim's program. To reset the vector:

```
POKE 145, 228
```

## Joystick/Keyboard Routine

The following is the Basic loader for Dave Hook's Joystick/Keyboard routine. The machine code source was listed but the Basic loader and demo program was accidentally omitted. Sorry Dave

```
0 REM JOYSTICK/KEYBOARD FOR ALL ROMS
1 DATA  32, 228, 255, 201,  48, 240,  15, 201,  32, 208,   2, 169
2 DATA  48,  56, 233,  48,  48,   4, 201,  10,  48,   2, 169,   5
3 DATA 168, 162, 255, 173, 198,   2,  16,   6, 142,   3,   2,  76
4 DATA 180,   2, 134, 151,  76, 180,   2, 173,  79, 232,  74,  74
5 DATA  74,  74,  45,  79, 232, 168, 185, 199,   2, 168, 169,   0
6 DATA 174, 198,   2,  16,   3,  76, 120, 210, 208,   3,  76, 109
7 DATA 210,  76, 188, 196,   0,   5,   5,   5,   0,   5,   7,   9
8 DATA   8,   5,   1,   3,   2,   5,   4,   6,   5
9 FOR I=634 TO 726 : READD : POKEI,D : NEXTI
10 POKE710, SGN(PEEK(50003)-1)
11 INPUT"USING JOYSTICKS   Y[CR CR CR]";A$
12 LO=165 : IF A$<>"Y" THEN LO=122
13 POKE1,LO : POKE2,2 : CLR
14 FOR I=1 TO 1000 : N=USR(0) : PRINTN; : FOR J=1 TO 50 : NEXTJ,I
```

## Programs to WordPro Conversion

This tidy little Basic program converts programs to WordPro files. It was written by Paul Higginbottom of Commodore Canada's Software Department.

The program does not convert special characters such as cursor graphics. These would have to be changed to CHR$( values or some other representation which would be unfeasible. Instead, these characters are left alone and show up in WordPro just as they do in a normal LIST. From here, you can use WordPro to edit them to suit (and what better editor). The program was actually used here to convert itself for publication. Then alpha characters were substituted for cursor characters that don't print on the NEC Spinwriter. (remember, "left-arrows" can't be printed from WordPro)

```
100 dim a$(90):for i=0 to 90:read a$(i):next
110 print "program filename ";:gosub 890:fi$=a$
112 print "wordpro filename ";:gosub 890:wf$=a$
115 input "device number      8[CR CR CR]";dv
120 open 2,dv,2,fi$+",p":gosub 880:get#2,a$,a$
121 open 3,dv,3,wf$+",p,w":gosub 880:print#3,
    chr$(192)chr$(91);
125 sl=0:get#2,a$,a$:if a$="" then 600:rem skip link un
    less end of program
126 print "[HM HM CLR]"chr$(14);:get#2,a$,b$:rem get line
    number
140 n=asc(a$+chr$(0))+asc(b$+chr$(0))*256:print n;
150 get#2,a$:p=asc(a$+chr$(0)):if p=0 then print "
    [left-arrow]":goto500
160 if (peek(205)<>0) or (p<128) then print chr$(p);:
    goto 200
170 printa$(p-128);
200 if (a$=":" or a$=",") and (peek(198)>45) then 220
201 if peek(198)>55 then 220
210 goto150
220 print "[left-arrow]":print n;chr$(150);:sl=sl+1:goto 150
500 for l=0 to sl:q=32768+l*80
505 for i=0 to 79:p=peek(q+i):print#3,chr$(p);:
    next
510 next
520 goto 125
600 close 3:close 2:end
880 if ds<20 then return
885 print ds$:end
890 poke 623,34:poke 624,27:poke 158,2:input a$:
    return
900 data end,for,next,data,input#,input,dim,read,
    let,goto,run,if,restore,gosub
910 data return,rem,stop,on,wait,load,save,verify,
    def,poke,print#,print,cont
920 data list,clr,cmd,sys,open,close,get,new,
    tab(,to,fn,spc(,then,not,step,+,-
930 data *,/,↑,and,or,>,=,<,sgn,int,abs,usr,fre,
    pos,sqr,rnd,log,exp,cos,sin
940 data tan,atn,peek,len,str$,val,asc,chr$,left$,
    right$,mid$,go,concat
950 data dopen,dclose,record,header,collect,backup,
    copy,append,dsave,dload
960 data catalog,rename,scratch,directory
```

Sieg Deleu,
Kobetek Systems

1. On concatenating files

On page 30 of the DOS manual it states that in the COPY command, the destination file name may be a new name, or the same as the old file name, unless the destination and source drives are the same.

When I first read this, I assumed that this meant that when concatenating files on the same drive, the destination file name had to be different from any of the names of the files being concatenated. This turns out not to be so, which is quite useful. If, for example, you have a sequential file (PERM) to which you keep adding data: you create a temporary file (TEMP) into which you write the data to be added to the PERM file. When finished, concatenate TEMP to PERM, and scratch TEMP.

2. On using variables in the disk commands.

I tend to use a standard set of subroutines for handling file operations such as opening, scratching, concatenating, etc... I don't remember having any problems with that under DOS 1.0. However, during a recent project I ran into some very annoying troubles, which caused me a good amount of headaches.

Consider the following two lines, part of a larger subroutine, to concatenate two files (assume command channel open on channel VC):

```
2000 DR$=STR$(DR):REM DR IS THE DRIVE BEING ACCESSED
2010 REM CONCATENATE XG$ TO XF$
2020 PRINT#VC,"C"+DR$+":"+XF$=DR$+":"+XF$+","+DR$+":"+XG$
```

Everything should work, right? It seems to me it used to! When you use these lines in your program, it will in fact run like a charm. We assume that you also have an error routine in your program, to check when anything goes wrong during file operations (check DS$). No error occurs when the above code is executed, but the concatenation operation did not take place! I found this out the hard way - I did not suspect THAT part of the code, and went looking everywhere else trying to find out why the data that was supposed to be in my file was'nt there. Being a neat person, and not wanting to clutter up my disk, I scratched XG$ as soon as the concatenation was done - vastly increasing the time necessary to trace this bug.

Solution: 2000 DR$=MID$(STR$(DR),2): REM to get rid of the space created by the STR$ function. DOS 2.0 detests spaces in disk commands.

3. On closing files.

Sometimes your program bombs with a syntax error just after you have opened a disk file. And sometimes the active light on the drive stays on, even when you issue a DCLOSE. You're a little worried about it, since you want to start checking your syntax error, and don't want anything to happen to your file in the mean time.

Solution: Type direct command - OPEN 1,8,15:CLOSE 1, and the disk goes to sleep.

4. Spaces in files.

We all know now that leading blanks in a record are lost when INPUTting from a disk or tape file. If you need blanks at the beginning of a record, use shifted ones (CHR$(160)) - the PET does'nt know they are blanks and gladly gives them back to you on INPUT. They PRINT on the screen as perfectly legal blanks of course.

5. Other nasties in files.

I am what you call a "squasher" of disk files. Whenever I get a chance, I'll "code" pieces of information so that they occupy as little space as possible. Example: suppose you have a series of Yes/No answers to be placed in a disk file. Let's say you have 8 of them, or less. You could use 8 bytes, one each with either a Y or a N. Or you can store this sequence of 8 answers in a single byte, by building a binary number out of the 8 answers, coding a Y as a 1, and a N as a 0. (Some people have been known to use the opposite convention - but "that's stupid", as Paul would declare). Below is a short piece of code to do it (we assume that the answers were input by your program into an array ANS$ - i.e. ANS$(K)="Y" or "N", for K from 1 to 8):

```
2000 ANS%=0
2010 FOR I = 1 TO 8: IF ANS$(I)="Y" THEN ANS%=ANS%+2**(8-I)
2020 NEXT I
```

The code constructs an 8-bit binary number that has a 1 in position i (leftmost position considered to be position 1) if ANS$(i)="Y", and a 0 in every other position. For example, NNNNYNYY will turn ANS% into 00001011, or decimal 11. We can now write this packed information to a file by PRINT# 1f, CHR$(ANS%). Is'nt that great? Well, it is, provided we can unpack it easily, and provided we take a little care when writing it to the file.

I don't particularly like to do GETs from long records, so I want to get the information back via an INPUT statement. Assume that this ANS% byte is part of a longer string record (REC$), e.g. 20 character name, 8 Y/N answers, # of children, # of bathrooms in the dwelling.

Before proceeding, we should note that the record as described can be stored in 23 bytes, as long as we don't have more than 255 children or bathrooms. 255 is the biggest binary number that can be held in one byte (binary 11111111). We PRINT#lfn, NAME$; CHR$(ANS%); CHR$(#children); CHR$(#bathrooms) to our tape or disk file.

To get it back, we:

```
100 INPUT #lf, REC$
110 NAME$=LEFT$(REC$,20): ANS%=ASC(MID$(REC$,21,1)
120 CHILD=ASC(MID$(REC$,22,1): BATH=MID$(REC$,1)
130 REM: NOW TO ISOLATE OUR Y/N ANSWERS INTO ANS$
140 FOR K=8 TO 1 STEP -1: QZ%=ANS%/2: QR%=ANS%-2*QZ%
150 ANS$(K)="Y": IF QR%=0 THEN ANS$(K)="N"
160 ANS%=QZ%: NEXT K
```

So far, so good. HOWEVER - what happens if all the answers were N, or if the answers were NNNNYYNY, or if we had no children or thirteen bathrooms? What's special about these? They cause our program to become very confused, and sometimes result in the programmer becoming equally stumped. In all four of the above instances the program will either write a CHR$(0) or a CHR$(13) to the file, and nicely terminate the record in the wrong place. Both of those CHR$'s are recognized by the PET as end-of-record markers, and prevent us from getting to the rest of the record which follows the CHR$. What do we do about that? We make sure that they cannot occur as part of the file, except where we want them.

A few suggestions:

a. Start ANS% with a value other than 0, and which does not occur as one of the actual Y/N combinations. This is not feasible in our case of 8 Y/N's, since the entire byte ANS% is needed to cover the whole range of possibilities (from 0 (00000000-all N's) to 255 (11111111-all Y's). In other words, there is no value with which ANS% can be initialized to avoid the "null or thirteen problem". Here we are forced to use GET instead of INPUT.

For sets of Y/N's of 7 or fewer, we can start ANS% at 2 to the power X, where X equals (# of Y/N answers):

Examples:

| Nr of Y/N's | Starting value of ANS% |
|---|---|
| 2 | 4 |
| 5 | 32 |
| 7 | 128 |

Let's look at 7 in more detail:

- if our answers are NNNNNNN, then ANS% will be 10000000 (binary), and CHR$(ANS%) cannot cause any problems when written to the file
- if our answers are NNNYYNY, ANS% equals 10001101, and again no record terminator is written.

On reading the file, we must of course remember that the value we read must now be reduced by the value we gave ANS% at the start. Line 110 in our short example will now read:

```
110 NAME$=LEFT$(REC$,20): ANS%=ASC(MID$(REC$,21,1)-INIT
111 REM INIT IS STARTING VALUE OF ANS%
```

b. In our children-bathrooms example, the same ideas are of course valid. If we have N bathrooms, where N<128, we can add 128 to N before writing it to the file, and then subtract it from the value we read back before we actually use it in our program.

c. There is still one small thing to watch out for. When testing our program, we usually like to read stuff that we have written to a file, and display it on the screen, to make sure that what we thought we wrote did in fact get written. Watch out when you do this if you have compacted some of your data using the technique mentioned above. Many of the CHR$ characters that you write are non-printing ones - they don't appear on the screen on a PRINT command - in fact, they look like nulls (if you print them between two markers, say two *'s, the *'s will be printed as **, i.e. with nothing in between). Now that's OK as long as you are aware of this, and you don't have a CBM8032.

That beauteous machine grabs some of these CHR$'s, and interprets them as control codes, to define a screen window for example, or to ring its chimes. I recently printed out a large file to browse through the records, looking for abnormalities. The screen was scrolling merrily upward as the records were displayed, occasionally squeaking as a chr$(7) was printed. Suddenly the screen contracted, and the display was confined to the rightmost 10 columns of the screen. This was sort of cute and I could still read the data, although I had to hurry, since the short lines zipped by at a good clip. The next thing was'nt cute at all: the screen collapsed to a single character, at the bottom right hand corner, and the rest of my file just winked at me from that spot as it flashed by - still ringing the funny little bell at times.

Moral: When you want to look at "squashed" files, GET the characters one by one, and print out their ASC values. Make sure that you change any nulls to CHR$(0) prior to printing on the screen, otherwise your program will hang with an ILLEGAL QUANTITY ERROR - ASC("") is a nyetnyet.

6. How full is an empty file.

Assume as I suggested in 1 that we want to add to a sequential file, using concatenation, every time a program is run.

```
10 OPEN 2,8,2,"1:TEMP,S,W"
20 REM STORE DATA TO IN TEMP
30 REM WHEN FINISHED, CONCATENATE WITH PERM
40 CLOSE 2
50 OPEN 1,8,15:PRINT#1,"C1:PERM=1:PERM,1:TEMP"
60 PRINT#1,"S1:TEMP":CLOSE 1:END
```

In order for line 30 not to abort with a FILE NOT FOUND error, PERM must exist prior to concatenation. It must therefore exist even before the program is run the first time. It seems natural therefore to issue a direct command:

```
OPEN 2,8,2,"1:PERM,S,W":CLOSE 2
```

This will put an empty file on the disk, so that the very first concatenation can properly take place. Right? - Well... almost.

The file will get on the disk; the very first concatenation will take place; but when you next want to read the PERM file to get your data back, there is a little too much in the file. This can throw things off a little or quite a lot, especially if you do GETs, where each character has a specific meaning, e.g. in a "squashed" file.

As it turns out, the "empty" file, created by the direct command, has not just one character in it, but a total of four! It has a CHR$(13), then a CHR$(0), a CHR$(2) and another CHR$(13). OK, now we know, so we just do four dummy GETs to skip past them when we read our PERM file after concatenation. No, we just do one dummy GET - after the first concatenation, the last three of the four extraneous characters vanish, and we're left with just one extra annoying carriage return (CHR(13)).

To prove it to yourself:

```
10 OPEN 3,8,3,"1:EMPTY FILE,S,W":CLOSE3
20 OPEN 3,8,3,"1:ONE CHAR,S,W":PRINT#3,"1":CLOSE3
30 OPEN1,8,15:PRINT#1,"C1:EMPTY FILE=1:EMPTY FILE,1:ONE CHAR"
40 OPEN 3,8,3,"1:EMPTY FILE,S,R"
50 GET#3,A$:IF ST=64 THEN ST%=-1
60 IF A$="" THEN A$=CHR$(0)
70 PRINT "#"A$"#"ASC(A$):IF NOT ST% THEN 50
80 CLOSE 3
90 PRINT#1,"S1:ONE CHAR":CLOSE 1:END
```

```
The output of this is
#          (carriage return printed)
# 13 #     (ASC of CR)
#1# 49 #   (character 1 and its ASC value)
#
#13#
```

Note that we wrote only two characters ( 1 and a carriage return ) to ONE CHAR, yet we end up with three in the concatenated file. The first carriage return is the leftover from opening the empty file.

Some (more) trivia?

If we time the sequence in lines 50 to 70, then we get 5.7 seconds. If we replace the name EMPTY FILE with FILE, i.e. a name different from the two being concatenated, then this same sequence takes 10 seconds - almost double!?!?!

The time to concatenate two files is a bare 1/4 of a second!!

7. The last word on nulls.

If you write a CHR$(0) to a file, and then GET it back (GET A$), then A$ does not equal CHR$(0), but rather comes back as a measly null character (""). This may not seem like much (eh!eh!), but it does become crucial when you want to copy one sequential file to another character by character - say because you processed part of it and now you want to discard that part and process only the remainder. During the copy, you must check if a character from the old file equals null("") and change it to CHR$(0) before writing it to the new file - otherwise you end up with a mess.

8. Time out error (variable ST=1)

I had never had one of those (at least not that I know of), and was therefore surprised and a bit pleased to finally get hit with one. Can you spot why in what follows? (FILE2 contains plenty of data)

```
10 OPEN 2,8,2,"1:FILE2,S,R":OPEN 3,8,3,"1:FILE3,S,W"
20 OPEN 1,8,15: REM DO A BUNCH OF STUFF
30 CLOSE 1
40 GET#2,A$:IF ST=64 THEN 60
50 PRINT#3,A$: GOTO40

Timeout occurred in line 50.
```

Editor's Note-

Timeout is an IEEE condition indicating that a bus operation has taken too long to complete. The time allowed is 64 milliseconds and timeout can occur during a read or write operation. In the above example a timeout on write occurs at line 50; an attempt to send A$ across the bus is made but the DOS does not accept the character within the 64 ms. time limit. PET sets ST is set to 1 but no test is made to trap this error here.

Leaving this for a moment, another disaster lies in the above example. By now we all know that the problem lies in the OPEN1,8,15:CLOSE1 sequence of lines 20 and 30. (Sorry if I revealled that too early Sieg) As mentioned earlier, this causes the DOS to close down any open files on the disk but the 8032 still considers these files open since no CLOSE 2 or CLOSE 3 command was given. The GET# statement in line 40 is still honoured (i.e. no FILE NOT OPEN ERROR) but the disk has nothing to offer since all files (in the DOS) are now closed. A timeout on read (ST=2) occurs but there is no Basic to detect this (only ST=64? is tested).

The program then goes on to send the "gotten" character but (besides the fact that there is no character in A$ to send) there is no open file in the disk to send it to. The DOS won't accept A$ and after 64 ms., timeout on write is flagged (ST=1).

None of this should be of any concern if you handle your I/O files properly. Opening and closing the DOS command

channel should not be used to close other R/W files except as mentioned earlier. By issuing proper OPEN and CLOSE commands, both computer and disk will always know what's happening, not to mention the programmer. Same goes for Basic 4.0 DOPEN and DCLOSE commands.

One last note, timeout on read (ST=2) will occur if you try reading past the end of a file. This usually happens when 'end of file' (ST=64) is tested after some subsequent bus operation is performed that changes ST. If this is a problem, the best solution is to "trap" ST into some other variable (say SX) and then test SX later.

Compressed Data

The above article talks about compressing data of the YES/NO, OFF/ON type so that several items can be stored as bits rather than using a whole byte for each. This can work well provided your program knows how to handle it. It can also save a lot of storage space when working with large amounts of information. The only drawback (aside from those mentioned) is that now your (user's) data is in unreadable by other software, i.e. compressed data can look pretty alien to a simple file reading program, especially when compared to what was thought to be typed in. Compressing data is generally undesirable, but if you find yourself cramped for space and absolutely require it, always provide plenty of documentation... if not for the next guy, at least for yourself!

## Jim Strassma's SUPERSORT

SUPERSORT is an extremely fast and powerful machine language sort for multi-dimensional memory-resident arrays. It has features which surpass those on computers many times the price and size of the Commodore line. It "lives somewhere" at the top of memory of ANY PET or CBM, in about 1200 bytes. Its exact location depends on what your computer already has at the top of its memory (DOS Support for example). SUPERSORT adjusts the memory pointers to make room for itself, and then protects itself, and everything above it in memory from being clobbered by BASIC. It then announces the SYS address by which to invoke it.

### FEATURES

1. Sorts one- and two-dimensional string and integer arrays at lightning speed, in adscending or descending order(e.g. 3000 integers in less than 30 secs!).
2. Records may have up to 75 fields (a 10 by 50 array for example stores 11 records of 51 fields each). The fields may be of random length, and require no special delimiters.
3. Sort may be specified on any one of the fields.
4. You may specify a subsort on any field (and on as many as desired in any order) if a match is found in the previous field.
5. The array to be sorted may be specified by name. If no name is given, SUPERSORT processes the first array in your program.
6. You may specify the sort on all or part only of an array. This allows you to set up a large enough array to sort the largest possible dataset, without running into re-dimensioning problems.
7. Provides a pattern-matching option, to allow you to divide your dataset into records that match/do not match a specified bit-pattern.
8. All options have convenient default settings. Desired changes in the default are provided through simple POKE statements.

This program is a GEM, and at the price of $ 45.00 Canadian on cassette ,including first class mail ($ 50.00 on disk), you cannot afford not to use it! It comes complete with clear instructions. Why so cheap? Don't ask, just get it - you will wonder how you ever got along without it!!

KOBETEK is the exclusive distributor of SUPERSORT in Canada. Dealer inquiries welcome - we offer an excellent deal!

## First Programming Steps.

Jim Butterfield, Toronto

The first programs that a beginner writes tend to be simple. That's good, of course: the programmer is developing skills which will be useful when he tackles more ambitious jobs. Here are a few suggestions on how to go about these early projects; the emphasis will be more on sound practices and clear style rather than clever coding methods. Some of the suggestions might be useful for experienced programmers, too...

Try to lay out your programs in "blocks". Each block should have a clear, simple function. One block might do an input job, another might calculate, and a third generate output. If you start planning a program by thinking out the blocks you will need, your program will be better planned. Some programmers make each block into a subroutine so that the main program simply calls in these units as needed.

Title each section or block with a remarks REM statement. You don't have to put comments on each line, but it's useful to be able to find a section of code quickly. Perhaps you think that you can remember the code - after all, you wrote it - but wait a couple of months. It's amazing how a crystal clear program can suddenly become gibberish after you've been away from it for a while. Leave yourself some highway markers so that you can find your way around later.

Name your variables in a semi-meaningful way. Totals can start with the letter T, counts with a C, and so on. I'm not a fan of large alphabetic names, since they have pitfalls: TERRIFIC is a great label, but it doesn't work since the keyword IF is hidden in the middle. Can you find the hidden keywords in GRANDPA, CATNIP, CRUNCH and FRONT? It's fun to play word games, but not when you're trying to write a program. I prefer a single letter followed by a numeric: T4, B7 and so on. By the way, don't forget that variable B has nothing to do with integer variable B% or string B$ or for that matter array variable B(3). They are all completely independent values.

Don't let anyone hustle you about program size or speed. If others write in less memory and fewer milliseconds, let them. You'll have space enough for most of your programs and the tenth of a second saved in run time won't give you time for a cup of coffee. On the other hand, do look for better methods. Better isn't always faster or smaller, but you'll recognize it when you see it.

Keep track of your variables; it's useful to make a list on a sheet of paper. That way, you won't accidentally use variable X for two different jobs and get them mixed up. In fact, it doesn't hurt to do paperwork planning before turning your computer on. There's a kind of "heat" in working directly on the machine that sometimes leads to hasty programming. A little leisurely planning beforehand can generate sounder and better programs.

Don't be afraid to write loosely. The fanatic who tells you that you'll save memory and time by compacting FOR M = S TO P STEP V into FORM=STOPSTEPV is steering you wrong in most cases. If legibility costs you four bytes and one millisecond, take it: it's a bargain.

If your program doesn't work right the first time, don't lose heart. It happens to most of us. The easy errors are where the computer tells you where the problem is, most commonly ?SYNTAX ERROR IN ... The problem will likely be obvious when you look at the line; if not, you can try rewriting it slightly to see what happens. The hard errors are where the computer doesn't stop, but gives you the wrong answers.

Debugging can be great fun if you take the right attitude. Look at the variables: you can call them up with direct PRINT statements. Change them if it suits your purpose. Put STOP commands into your program and check everything out when you come to the halt. You can resume where you left off with CONT. Using the RUN/STOP key to break your program in mid-execution is less precise but will also do the job.

Getting a program together can be a rewarding experience - not necessarily rewarding in money, but in a sense of accomplishment. Each program will be a work ofart, done in your own style. When you put your signature to your latest masterpiece, you'll feel good about it if you've used good coding craftmanship.

One of the friendliest things about the PET, CBM and VIC is the way they allow you to make a change or correction. If the line on the screen is wrong - whether it is a program line or a direct command - we can move the cursor back and type over the line. Pressing the RETURN key will make the change take effect.

Correcting Programs

This is very handy for programs. When your first program attempts result in a message such as ?SYNTAX ERROR IN 350 you can list 350 to see what the trouble. If line 350 happens to say PWINT X, you can move the cursor back, type R over the W to give PRINT X, and strike RETURN. The line has been corrected with a minimum of typing on your part.

If you need to make an insertion into your program, you may use the INSERT key. If the mistake was PINT X, the technique is to position the cursor over the I, hold down the SHIFT key, and press INST for insert; the computer will open up space and you can type in the missing R. On the other hand, if the error was PHRINT X you'll want to make a deletion: place the cursor over the R, press the DEL key to delete, and the H will disappear. In either case, don't forget to press RETURN to make the change permanent.

If you happen to goof in making the change, start over. In this case, don't press RETURN. Hold down SHIFT and then press RETURN: this will take you to the next line without any program change being made.

Shifted-RETURN is quite a handy key combination to know for many reasons. If you wanted to leave a note on the computer's screen for someone to read, you might type MARY - PUT THE CAT OUT. At this point, striking RETURN would cause the computer to try to "perform" the line, and you'd get ?SYNTAX ERROR. If you press Shifted-RETURN, however, you'll just go to the next line and the computer won't try do anything with the contents of the previous line.

The INSERT key has some special rules. After you have pressed the INSERT key a number of times (don't forget to hold down SHIFT) there will be an open space on the screen where you can insert the new characters. At this point, you'll be in "programmed cursor" mode. This means that the cursor keys don't move the cursor; instead they will print as special reversed characters. This is the same way that the PET behaves after you press the quote-mark key, with two important exceptions: the computer remains in this mode only for the number of characters to be inserted; and the Delete (DEL) and Insert (INST) keys work in a different way. More about this another time; in the meantime, you'll get used to them quite quickly.

A problem sometimes crops up if a program line is too long. Sometimes this means that there's no extra space available to make a desired insertion - eighty characters is the screen limit. Worse, the line is too long to start with; it occupies over 80 characters even before we make a change. It might be more sensible to change it to two lines and relieve the crowding; but if you must, the trick is to look through the line to find a keyword that can be abbreviated. PRINT is the most popular, since it can be rewritten as a question mark. Close up the space, making sure that everything is packed into the 80-column work area, and then make the change if it fits.

The Direct Approach

Direct lines - Basic commands typed in without a line number so they are executed right away - are usually easy to fix. If you mistype LOAD "PROGRAM" so that it comes out LOUD "PROGRAM", don't be dismayed by the ?SYNTAX ERROR. You can slip the cursor back, change the U to an A, press RETURN and the load will take place.

Correcting mistakes in Direct lines can leave a cluttered screen. When I try to load BOTTLESHIPS from the disk, I get several lines which tell me that there's no such program. When I move the cursor back and correct to BATTLESHIPS, the following lines don't go away unless written over. It looks messy, but works OK.

There's a sharper problem when I ask a direct statement to print a number. If I ask the PET to calculate 4*5*6, yielding a product of 120, and then decide that I really want addition, I can go back and change the asterisks to plus signs. The PET will now produce a total of 15, but the last digit of the previous answer won't be wiped out; the zero will be left on the screen and our sum will look like 150 instead of 15. The solution? Wipe out any numbers you want recalculated so that the new values will print on a fresh line.

Special Screenings

When you press RETURN, the PET sees only what's on the screen. You may have done deletions, insertions, and changes but the final screen result is all that counts. This is true of program lines, direct commands, and responses to program INPUT statements.

You may want to run a program several times while testing, with similar answers to INPUT questions on each run. With screen editing, it's a snap. After you have run once, move the cursor back to the RUN statement. Press RETURN (no need to type RUN: it's on the screen). For each INPUT, the cursor will appear over the answer you typed on the previous run. If you want to go with the same response this time, just

press RETURN and the program will accept the same input from the screen. If you want to change, type your new input.

Here's a hint of advance techniques that you'll learn as you become more familiar with your computer. You can actually get the PET to type its own input - even its own program changes - to the screen. Then, with a stroke of the RETURN key, you can activate the input or program change. When used for INPUT activities, this provides a "default" input for the user. As a program change, the program could suggest DATA statements that it would like to see included in a future run. Mind boggling! At this rate, the computer could program itself and make us all obsolete.

At least, the computer still needs us to press the RETURN key; it can't do that by itself. Or can it? Technical tyros suggest that POKE 158,1:POKE 623,13 (or on Original ROMs, POKE 525,1:POKE 527,13) would actually cause the PET to send a carriage return to itself...

SWARM-100 -- SWAP A ROM MODULE FOR PET/CBM — David Hook, Barrie Ont.

Producer: Software Unlimited, Oakville, Ontario, CANADA
Designer: Dieter Demmer
Cost: $150 (Canadian)
Availability: Batteries Included, 71 McCaul Street, TORONTO, Ontario, CANADA, M5T 2X1

The SWARM-100 is an 8" x 4" printed-cicuit board. It installs completely inside the PET. There are two rows of seven sockets for installation of two independent operating systems. Selection of either system is done via software. The swapping of ROMs may be accomplished without loss of the program in memory.

The SWARM-100 is a solution for PET owners with Basic 2.0 who want Basic 4.0 also. Many commercial programs won't function with the new Basic; many programmers won't be able to convert machine language themselves.

This past week I received a letter from the University of Waterloo advising that they would be happy to provide me with an upgrade EPROM for my $61.50 Waterloo Basic chip. With a price tag of $35 for this service, I'll politely ignore the offer. The SWARM board frees me from such outrageous costs.

Since the documentation refers only to "new" Pets, I'll presume that the board is suitable for 2001 or 4000 series machines. We've been told elsewhere that the "original" Pets may only be upgraded to Basic 2.0.

The Toronto PUG has over 1400 programs in its library. I've felt the need for both Basic 2.0 and 4.0 in the screening of contributed programs. My 2.0 machine was purchased prior to the "swap" offer from Commodore. Therefore, when I purchased the upgrade ROMs, I had a set of 2.0 left over.

Installation:

The instructions provide a complete description and diagrams of the proper arrangement. Read them carefully if you are trying it yourself.

Don't try to extract the ROMs with a screwdriver, as suggested. A cheap (under $1) IC-puller is widely available. Bent or broken pins may not be salvageable!

The diagram does not clearly indicate that Row 1 is the BACK row (when holding the board with sockets on your left). Row 2 (FRONT) is where to place the Basic 4.0 ROM set. This is the row that is alive on power-up.

The pins on the underside of the SWARM are quite sturdy. They are wire-wrap pins and will likely enlarge the PET's sockets. There are 33 pins to force into the PET sockets. Removing the entire main logic board is highly recommended. I know that I was quite leery of the force required to fit them snugly. Perhaps the dealer's expertise is the best approach for the uninitiated (like myself).

Operation:

Mr. Demmer has used three "non-existent" memory addresses to control the operation of the SWARM.

Basic 4.0 requires five of the seven available sockets. These five in both rows are switchable. Memory addresses range from $B000 through $FFFF. These will be referred to as the SYSTEM ROMs.

The two pairs of sockets covering $9000 and $A000 are the UTILITY sockets. They may be switched quite independently of the System ROMs. This allows you to have both WordPro3/BPI or Waterloo Basic/Jinsam available at a moment's "swap".

The 4-page manual provided gives a short machine language routine demonstrating the software control of swapping the System ROMs. Basic 2.0 and 4.0 have different locations for the interrupts that occur 60 times per second. Machine language is necessary to safely make the various swaps possible.

The program shows how to implement a swap from 2.0 to 4.0 and vice versa. The proper code for cold-start (reset) and warm-start (no loss of program) is given.

You may use a Basic POKE command to swap the Utility row from front (4.0) to back (2.0) row. If you are using Basic 4.0, another POKE gets the Utilities back to the front row.

When the System is in Basic 2.0, the above POKE will swap the Utility pair to the back row. But machine language is the only way to get these two back to the front row.

A Better Program?

I've put together an all-in-one program to give maximum control of the available options. It's in the form of a Basic loader which will permit storage:

1. Cassette#2 buffer, but safe from Basic 4.0 usage.
2. Top of memory, moving down the appropriate pointers.
3. Anywhere in RAM, user-specified, either in decimal or hex.

The loader is self-relocating, adjusting the internal memory reference automatically.

After RUNning three "SYS" addresses are displayed:

1. SOFT: swap either way, retaining program in memory.
2. HARD: swap either way, with a power-on reset.
3. UTIL: allows restoring of Utility ROMs to front row when in Basic 2.0

Copy the addresses from the screen. Referring to the above program here is a table of options:

| Action | | In Basic 4.0 | In Basic 2.0 |
|---|---|---|---|
| Utility | -- front/back | POKE 59444,0 | POKE 59444,0 |
| | -- back/front | POKE 59452,0 | SYS(UTIL) |
| System | -- keep program | SYS(SOFT) | SYS(SOFT) |
| | -- reset PET | SYS(HARD) | SYS(HARD) |

A Quirk:

When swapping System ROMs in a "soft" fashion, there are some Basic pointers that are left in an "unknown state". Mr. Demmer advises that this should not pose a problem in RUNning the program. In my brief experience, no problems have developed.

If you attempt to modify the program after the swap, you may find a "reluctant" cursor. When 'RETURN' is struck, instead of advancing to the start of the next line, the cursor may sit in mid-line.

If you've used CMD to print a program listing, then immediately CLOSE the file, you've seen this animal before. (Most everyone knows to do a 'PRINT#' before doing the CLOSE).

No real problems here, either. Mr. Demmer suggested that you can generate a SYNTAX ERROR to restore normal behaviour. Do this with two consecutive double-quotes ("") and 'RETURN' and you're home free.

Summary:

The SWARM-100 is a well-designed, good quality product. It fits inside the PET/CBM and permits software selection of either of two operating systems. The two pair of spare sockets may be swapped independently of the other pair of five sockets.

The product has a three-month warranty for parts and labour. It must be returned to the dealer for repair.

If you need both Basic 2.0 and Basic 4.0 in your machine, the SWARM-100 does the job.

```
LINE# LOC   CODE        LINE

0001  0000              ;*****************************************
0002  0000              ;*                                       *
0003  0000              ;* ROMSWITCH & UTILSWITCH FOR SWARM-100 *
0004  0000              ;*                                       *
0005  0000              ;*    DAVID A. HOOK, 58 STEEL STREET     *
0006  0000              ;*       BARRIE, ONTARIO,  CANADA        *
0007  0000              ;*     L4M 2E9       (705) 726-8126      *
0008  0000              ;*              MAY 17, 1981             *
0009  0000              ;*                                       *
0010  0000              ;*****************************************
0011  0000              ;
0012  0000              ;***  BASIC VARIABLES
0013  0000              ;
0014  0000              IRQ=$90                 ;IRQ VECTOR
0015  0000              ;
0016  0000              ; VECTORS FOR BASIC 4.0
0017  0000              ;
0018  0000              INVEC4=$E455            ;NORMAL INTERRUPT
0019  0000              BRKVC4=$D478            ;BREAK
0020  0000              NMIVC4=$B3FF            ;NMI (READY)
0021  0000              ;
0022  0000              ; VECTORS FOR BASIC 2.0
0023  0000              ;
0024  0000              INVEC2=$E62E            ;NORMAL INTERRUPT
0025  0000              BRKVC2=$FD17            ;BREAK
0026  0000              NMIVC2=$C389            ;NMI (READY)
0027  0000              ;
0028  0000              SWAP42=$E838            ;SWAP ROMS
0029  0000              SWAP24=$E83C            ;RESTORE ROMS/UTIL
0030  0000              ;
0031  0000              ;*** OP SYSTEM ROUTINES
0032  0000              ;
0033  0000              NMIVEC=$FFFA
0034  0000              RSETVC=$FFFC
0035  0000              ;
0036  0000                    *=$0381           ;SECOND CASSETTE BUFFER
0037  0381              ;
0038  0381  A2 05       SOFT  LDX #05           ;SET INDEX COUNTER
0039  0383  D0 02             BNE SOFT1        ;ALWAYS
0040  0385              ;
0041  0385  A2 00       HARD  LDX #00           ;IF HARD SWAP
0042  0387  A0 04       SOFT1 LDY #$04          ;OFFSET FOR 2.0 TO 4.0
0043  0389  A5 90             LDA IRQ
0044  038B  C9 55             CMP #<INVEC4     ;ARE WE USING 4.0
0045  038D  D0 02             BNE NOW2         ;NO
0046  038F              ;
0047  038F  A0 00             LDY #$00         ;OFFSET FOR 4.0 TO 2.0
0048  0391              ;
0049  0391  78         NOW2  SEI               ;DISABLE INTERRUPT
0050  0392  99 38 E8         STA SWAP42,Y     ;SET ADDRESS
0051  0395              ;
0052  0395  8A               TXA              ;CHECK IF SOFT SWAP
0053  0396  D0 03             BNE SOFT2        ;YES
```

```
LINE# LOC   CODE        LINE

0054  0398                ;
0055  0398  6C FC FF      JMP (RSETVC)    ;COLD START WITH SWAPPED ROMS
0056  039B                ;
0057  039B  98            SOFT2 TYA        ;GET OFFSET INDEX
0058  039C  0A                  ASL A      ;DOUBLE IT
0059  039D  A8                  TAY        ;KEEP IT IN R(Y)
0060  039E                ;
0061  039E  B9 AB 03      LOOP  LDA TABLE,Y  ;GET VECTORS IN REVERSE ORDER
0062  03A1  95 90                STA IRQ,X
0063  03A3  C8                  INY
0064  03A4  CA                  DEX
0065  03A5  10 F7               BPL LOOP
0066  03A7  58                  CLI
0067  03A8  6C FA FF            JMP (NMIVEC)  ;RETURN THROUGH WARM START
0068  03AB                ;
0069  03AB  C3 89        TABLE .DBYTE NMIVC2, BRKVC2, INVEC2, $0000
0069  03AD  FD 17
0069  03AF  E6 2E
0069  03B1  00 00
0070  03B3  B3 FF               .DBYTE NMIVC4, BRKVC4, INVEC4
0070  03B5  D4 78
0070  03B7  E4 55
0071  03B9                ;
0072  03B9                ;
0073  03B9  78            UTIL  SEI        ;SWAP UTILITY ROM TO 4.0 ROW
0074  03BA  8D 3C E8            STA SWAP24  ;RESET ALL TO 4.0
0075  03BD  8D 38 E8            STA SWAP42  ;SWAP ROMS TO 2.0
0076  03C0  58                  CLI
0077  03C1  60                  RTS
0078  03C2                ;
0079  03C2                      .END

ERRORS = 0000


SYMBOL TABLE


SYMBOL VALUE

 BRKVC2   FD17    BRKVC4   D478    HARD     0385    INVEC2   E62E
 INVEC4   E455    IRQ      0090    LOOP     039E    NMIVC2   C389
 NMIVC4   B3FF    NMIVEC   FFFA    NOW2     0391    RSETVC   FFFC
 SOFT     0381    SOFT1    0387    SOFT2    039B    SWAP24   E83C
 SWAP42   E838    TABLE    03AB    UTIL     03B9

END OF ASSEMBLY
```

## Bulletin Boards

Gord Campbell, Toronto

A new phenomenon has appeared in the Canadian small computer world. Several computer bulletin-board systems have become available in the last few months. If you have a 300-baud modem, (such as the Commodore 8010) it is easy to use these systems.

The main function they provide is implied by the name: the ability to enter and retreive messages. This helps people who have announcements to make or who wish to browse what others have to say. They are also a good place to ask questions, since someone is bound to want to 'be the expert'.

There are several systems, with a wide range of capabilities, but all of them have a common set of simple commands:

HELP - lists the available commands
SUMMARY - gives a summary of messages on the system.
RETREIVE - lets you look at specific messages
ENTER - to place a message on the system
GOODBYE - to disconnect tidily

In addition, several systems let you scan the messages on the system (command ALL) or list the users (LOG). A couple allow program upload/download, but require that you have appropriate terminal software. Most systems operate with single-letter commands for ease of use.

Here are the systems of which I am aware:

- The PSI/Wordpro Bulletin Board operated by Steve Punter. This is a very well-developed system. It operates on a PET with a modified Commodore modem. The program is a mix of BASIC and machine-language. Program upload/download is supported, with a special terminal program which Steve has developed. There are a number of commands which support program-swap, such as LIST, which prints out what programs are available. As a result, some of the commands must be spelled out in full. When a message is deleted, the others are renumbered, so it is unwise to refer to another message by number. By the same token, if the highest message number is 57, for example, you know that there are messages numbered 1 to 57 on the system. During message entry, it is not neccesary to press RETURN except at the end of a paragraph, since the program reformats your text into 38-character lines. The phone number is (416) 624-5431, with availability during non-business hours.

- The Toronto Pet Users Group bulletin board. This is a slightly more primitive system. It is written entirely in BASIC, so some people find they have to type a bit slower than normal. It operates on a PET with a TNW-103 modem. The fundamental functions are provided, as well as a list of other systems, which may not be all that current. The phone number is (416) 923-1917. Availability is during non-store hours.

- Remote CP/M operated by Jud Newell. This operates on a large S100 system in Mississauga. The software is extremely well developed. For example, when you sign on, it will tell you about any messages which are for you. As well, once you enter a message, you can edit it before saving. Program swap is supported for people running CP/M, using a program called XMODEM. You can also exit to CP/M, and operate as the main console of the system. This is not as powerful as it sounds, since there doesn't seem to be any BASIC compilers or interpreters available to terminal users. However, there is a lot of stuff stored on a hard-disk which you can browse. The phone number is (416) 826-5394, and the system is available nearly 24 hours a day.

- Apple Bulletin Board System. This provides the basic bulletin-board functions. The phone number is (416) 499-2908, and the system is available during non-business hours.

- Burlington Bulletin Board. I have not signed on to this, but have browsed messages describing it. Phone (416) 639-7209 days and weekends.

- Thunder Bay BBS. I have not used this system. It is apparently available evenings and weekends on an experimental basis. Phone number is (807) 345-7336.

All of the above information is, of course, subject to change. When you are offering a free service, you get to make your own rules as you go.

I am told that there are also systems available in Vancouver, Ottawa, Oakville, and perhaps Nova Scotia.

The three busiest systems described above have already had over 1500 callers in total. Try them out, and see what all those people are saying.

# Calculator Revisited

Morley E. Kipp
Mississauga, Ont.

In the October, 1978, issue of The Transactor, a suggestion was made for a program to simulate a calculator on the PET. A "starter" program was provided and a proposal put forth that it should be built upon and submitted for a follow-up article. Well, it kind of caught my fancy, and I fooled around with it for a while. Then I got involved with a few other things and forgot it. While browsing through some disks the other day, I came across it and realized that I had not seen any follow up in our favourite newsletter, so thought I would toss it in. It may be just the thing to convince your good lady that a computer can indeed be useful. (On the other hand, she may figure it's the world's most expensive calculator.)

It started in life as a fairly plain device, utilizing some of the built-in PET functions. It now has ten memories and is completely programmable, with the capability to save and recall programs. As presented here, it saves on disk, but that, of course would be quite easy to change to tape operation. It is straightforward to operate, so intructions will be on the sketchy side, in the interests of space conservation. The program will operate on both 40 and 80 column machines.

## Calculator Operation

Normal calculator operation is carried out on the PET number pad, with the exception of 'clear display' which is done with the left-arrow key. The more exotic functions are called up on the keyboard:

Q - Square Root
L - Logarithm
S - Sine
C - Cosine
E - Exponent
T - Tangent
P - Programmable Mode

## Programmable Mode

After hitting 'P' and entering Programmable Mode, the procedure is as follows: (All 'Program' transactions are entered in the screen space to the right of the calculator mock-up, except the actual run, which is displayed in the calculator window.) First, the prompt appears: 'NEW OR EXISTING?'.

### a) New Program

If you are writing a new program, the response to the initial prompt will be 'N'. The prompt, 'ENTER PROGRAM' will appear followed by step numbers starting at 1. Enter each number or function the same way as with any other programmable calculator, following each entry with the

'RETURN' key to register it. The last step of the program should be an 'equals key' (=) and the last entry of all MUST be the word 'END' which is the key for the program at all steps of its existence - loading from disk and running.

b) Existing Program

If the program exists in memory, you may run or modify it. If it has been previously saved on disk, you may recall it for re-running or modification. In either case, enter 'E' and the next prompt will be 'LOAD, SAVE, CHECK, RUN OR EXIT' (1, 2, 3, 4, or 5). Actions will be as follows depending on your response:

1) LOAD. You will be asked for the name of the program to be loaded. Respond with the name given to the file when you saved it on disk. The program will load and the last prompt will reappear.

2) SAVE. Here you will be asked to provide a name for the program. It will then be saved on disk in drive 0, and the prompt will reappear. If the program has not been previously saved but has been written during the current session, you would still respond with 'E' for existing, but would not tell the program to 'LOAD' when asked. Rather, you would 'RUN' or 'CHECK' it.

3) CHECK. The program must be in memory for this to work. It will be presented step by step. You must hit 'RETURN' as each step is presented, to re-enter it, or you may change the item to a new value or instruction. After 'END' is entered, the prompt reappears.

4) RUN. As the command implies, this initiates the run of the program in memory. Completion of a program run leaves you in standard calculator mode, with the program result displayed.

5) EXIT. Exit from programmable mode to calculator mode.

Memories

Handling of memories has been implemented as far as possible to conform with normal calculator usage. The program supports 10 memories; 0 - 9.

To Write a Memory

First display the number in the calculator window. To enter the displayed number into memory 1 as a positive number, hit 'W1+' ('WRITE memory ONE - ADD'). This action will also add the value in the display to any number already in the addressed memory. Similarly, entering 'W1-' subtracts the value in the display from the value in memory 1. To erase a memory, enter 'W10'. (WRITE memory ONE with ZERO).

To Read a Memory

To read memory 1, enter 'R1'. The value in memory 1 will be put into the display.

You may also read or write memories from program mode. Simply enter the individual instructions as one step each. That is, to write to memory nine, for example:

```
... STEP 5:   'W'  <RETURN>
    STEP 6:   '9'  <RETURN>
    STEP 7:   '+'  <RETURN> ...
```

I hope you enjoy using the program. It really is kinda fun, and may help put computers into a context that non-computer people can relate to, during a demonstration. (As in "But what can it DO???") The program listing follows and a copy will also be submitted to the Toronto PET Users Group library.

Editor's Note

Due to the amount of graphics in Morley's program, we chose to list it on an 4022 printer. As you can see, the program is quite neat and tidy, which means new features would be no trouble to add. One possibility might be a 'halt' function to allow programs to display intermediate results before continuing with the rest of the program. Another might be a stack implementation to allow a hierarchy order of operation (ie. programmable brackets). A LIST function could be implemented by simply modifying the SAVE command. Whatever you decide, I'm sure we all agree that Morley's program is a best start!

One final note, I beleive programs are stored in the A$ array. Line 140 would suggest a maximum of 150 steps but checking FRE(0) would indicate room for about 6000 more!

```
100 REM          ***  CALCULATOR  ***
110 REM FROM THE TRANSACTOR, OCTOBER, 1978
120 REM       MODIFIED BY MORLEY KIPP
130 REM            MISSISSAUGA ONT.
140 DIMA$(150):POKE59468,12
150 PRINT"▯      ┌──────────────────┐"
160 PRINT" 5 spaces |        17 spaces        |"
170 PRINT"      |                      |"
180 PRINT"      └──────────────────┘"
190 FORI=1TO19
200 PRINT"      ▮▮                    "
210 NEXT
220 IFR=1ORR=2THEN1030
230 REM CONTROLLER/INPUT
240 GETA$:IFA$=""THEN240
250 A=ASC(A$)
260 IFA>57THEN420
270 IFA<48THENIFA<>46THEN320
280 IFT=1THENX$=" ":T=0
290 IFLEN(X$)>9THEND$="▮▮ ERROR 5 spaces ":GOTO630:T=1:GOTO230
300 X$=X$+A$:X=VAL(X$):GOTO570
310 GOTO230
320 REM    OPERATORS
330 IFA<40ORA=44THEND$="▮▮▮ERROR     ":GOTO630:CLR:GOTO230
340 IFA=40THENN=N+1:B(N)=X:X=0:Y=0:O$(N)=O$:O$="":T=1:GOTO550:GOTO230
350 IFO$="*"THENX=X*Y
360 IFO$="/"THENX=Y/X
370 IFO$="+"THENX=X+Y
380 IFO$="-"THENX=Y-X
390 Y=X:O$=A$:T=1
400 IFA=41THENY=B(N):O$=O$(N):N=N-1:T=0
410 GOTO550:GOTO230
420 IFA$="S"THENX=SIN(X)
430 IFA$="C"THENX=COS(X)
440 IFA$="T"THENX=TAN(X)
450 IFA$="L"THENX=LOG(X)
460 IFA$="E"THENX=EXP(X)
470 IFA$="Q"THENX=SQR(X)
480 IFA$="="THEN350
490 IFA$="←"THENX=0:Y=0:T=1:D$="▮▮     16 spaces          ":GOTO630
500 IFA$="W"THENGOSUB670
510 IFA$="R"THENGOSUB790
520 IFA$="P"THENP$="P":GOTO840
530 GOTO550:IFR=4THEN1010
540 GOTO230
550 REM DISPLAY
560 X$=STR$(X)
570 G$="      15 spaces      "+X$:D$=RIGHT$(G$,11)+" 4 spa  "
580 IFABS(X)<=999999999ANDABS(X)>.01THEN630
590 IFX=0THEN630
600 IFABS(X)>1E38ORABS(X)<1E-38THEND$="▮▮▮ERROR 4 spa  ":GOTO630
610 G$="▮▮▮▮▮▮▮▮▮▮▮"+X$:R$=RIGHT$(G$,15)
620 D$=LEFT$(R$,11)+" "+RIGHT$(R$,3)
630 PRINT"▮▮▮▮▮▮▮▮▮▮"D$
640 IFR=4THEN1000
650 GOTO230
```

```
660 REM WRITE TO MEMORY
670 IFR=4THENA$=A$(I):GOTO690
680 GETA$:IF A$=""THEN680
690 IFASC(A$)<58THEN700
700 Z=VAL(A$):     IFR=4THENI=I+1:A$=A$(I):GOTO720
710 GETA$:IFA$=""THEN710
720 IFA$="+"THENM(Z)=M(Z)+X
730 IFA$="-"THENM(Z)=M(Z)-X
740 IFA$="0"THENM(Z)=0
750 TA$="■■■■■■■■■■■■■■■■":IFZ=0THEN770
760 FORT=1TOZ:TA$="■"+TA$:NEXT
770 PRINTTA$;"  15 Spaces          ■■■■■■■■■■■■■■■■■■";
780 PRINTZ"■■■■M■■=■■"M(Z):RETURN
790 IFA$(I)="R"THENA$=A$(I+1):I=I+2:GOTO820
800 GETA$:IFA$=""THEN800
810 IFASC(A$)>57THEN800
820 X=M(VAL(A$)):A$="":RETURN
830 REM     PROGRAM SECTION
840 PRINT"■■■";TAB(24);"NEW OR EXISTING":PRINTTAB(24);"   15 Spaces          "
850 GETE$:IFE$=""THEN850
860 IFLEFT$(E$,1)="E"THEN1030
870 IFLEFT$(E$,1)="N"THENV=0:GOTO890
880 GOTO840
890 PRINT"■■■";TAB(24);"ENTER PROGRAM   "
900 V=V+1:PRINT"■■■■■■■■■■■■■■■■■■■■";V;" 9Spc      ■■■■■■■■■■";:INPUTA$(V)
910 IFA$(V)="END"THENP$="P":R=0:GOTO1030
920 GOTO890
930 I=1
940 J=1
950 IFA$(I)="END"THENP$="":R=0:GOTO230
960 IFA$(I)="R"THENGOSUB790:J=1:GOTO980
970 IFA$(I)="W"THENI=I+1:GOSUB670
980 A$=MID$(A$(I),J,1):A=ASC(A$)
990 GOTO260
1000 IFJ<LEN(A$(I))THENJ=J+1:GOTO950
1010 I=I+1:GOTO940
1020 A$(V)="":GOTO230
1030 PRINT"■■■";TAB(24);"LOAD, SAVE,     "
1040 PRINTTAB(24);"CHECK, RUN,     "
1050 PRINTTAB(24);"OR EXIT         "
1060 PRINTTAB(24);"1,2,3,4,5       "
1070 PRINTTAB(24);:PRINT"   11 Spc      ■■■■■■■■■■";
1080 GETR:IFR=0THEN1080
1090 PRINT"■■■";TAB(24);"      15 Spaces      "
1100 PRINT"■■■■";TAB(24);"                    "
1110 PRINT"■■■■■";TAB(24);"                     "
1120 PRINT"■■■■■■";TAB(24);"                      "
1130 IFR<1ORR>5THEN1030
1140 ONRGOTO1160,1220,1280,930,230
1150 REM     READ PGM FROM DISK
1160 PRINT"■■■"TAB(24);"PGM NAME"
1170 PRINTTAB(24);:INPUTPM$:FI$="0:"+PM$+CHR$(44)+"S"+CHR$(44)+"R"
1180 OPEN1,8,2,FI$:I=0
1190 I=I+1:INPUT#1,A$(I):IFLEFT$(A$(I),3)="END"THENCLOSE1:GOTO220
1200 GOTO1190
1210 REM     WRITE PGM TO DISK
1220 PRINT"■■■"TAB(24)"NAME PGM"
1230 PRINTTAB(24);:INPUTPM$:FI$="@0:"+PM$+CHR$(44)+"S"+CHR$(44)+"W"
1240 OPEN1,8,2,FI$:I=0
1250 I=I+1:PRINT#1,A$(I);CHR$(13);:IFA$(I)="END"THENCLOSE1:GOTO150
```

```
1270 REM     PROGRAM CHECK AND RE-ENTER
1280 I=1
1290 PRINT"S████████████████████████████";
1300 PRINTI;"   11 Spc       ███████████";A$(I)
1310 PRINT"S██████████████████████████████";
1320 IFI>9THENPRINT"█";:IFI>99THENPRINT"█";
1330 PRINT" ";:INPUTA$(I):IFA$(I)<>"END"THENI=I+1:GOTO1290
1340 V=I:GOTO1030
```

## Spell Checking Programs, An Overview

Stew Martin
Mississauga, Ont.

Letting your wordprocessor correct spelling errors seems like one of the super great ideas of all time. Spell check programs are not particularly new to the industry. IBM has had them available as options on their big systems for quite a few years. Now, spell checkers are starting to attract the interests of WordPro users, and the first WordPro compatible spell check programs are beginning to appear on the market.

### The Problems

Consider this. The average college dictionary contains about 100,000 to 150,000 words, and Webster's New Collegiate Dictionary even boasts "...more than 22,000 new words...". How many bytes of memory would all those words take, I wonder. The average english word contains about 7 letters, or, in this case bytes. A little alpha-crunching could bring that down to about 5 bytes. Even crunched, we would need over a megabyte just for Webster's alone, without any room left for the spell checker program.

On top of all those words come the jargon and slang words developed by the inhabitants of specialized industries and vocations. An example recognized by legal people would be the word "tortious". Looks like a typo or an error to the rest of us, but it is a properly spelled legal term that does not appear in Collins. Engineers, teachers, medical people and legal advisors all have their own special jargon words, some of which will not appear in any dictionary.

Then there are the place names and proper names that we use in letters and documents every day. "Mississauga" isn't likely to show up in this years Collins Gem English Dictionary. Neither is Kim or Joan or Stephen. Starting to get the picture? First, ...there are a heck of a lot of words in a dictionary. Second, ...there are a heck of a lot of words that are <u>not</u> in a dictionary, and third, when a spell check program can't find a word in its dictionary files it assumes an error, and refers the suspect word to the operator for the final decision.

## Prefixes And Suffixes

How on earth are they going to jam a hundred thousand word dictionary into the 32K memory available in a Commodore 8032 or 4032 computer. Well, first of all, a reasonably good pocket dictionary has only about 30,000 or so references. Then, there are a few tricks and compromises that can be accomplished by using "base" words. A great number of everyday english words will conveniently break down into "base" words to which may be added common prefixes and suffixes. For instance, the word UN-CLAIM-ED. Using the same prefix and suffix and different base words we get UN-WANT-ED, UN-FOUND-ED, UN-OPEN-ED and so forth, effectively increasing the number of words from one base by a factor of three.

Consider all the english words that end with the letter "e". If we handle that ending "e" as a suffix, we can save a whole lot of dictionary space. Also, the base word WRIT-E will work with the suffixes WRIT-ER, WRIT-ING, WRIT-TEN. Unfortunately, it doesn't work in all cases. WROTE would have to stand as a base word on it's own. By various ways and methods that will remain undisclosed here, the spell check programs take about 2000 base words, allow the user to add 900 or so of his own commonly used words and special jargon, and very efficiently convert them into an effective 10,000 word dictionary.

## Compromises

Without question, the method is a compromise and you can see that the suffix -ER would be accepted as correct even if it showed up attached to the word DISK-ER, or LEAST-ER, or VARIOUS-ER which even I know is wrong. However, the argument can be made that the foregoing examples are not typical or likely spelling errors or typos, and can therefore be lived with.

Numerics can't be checked. If you write 1979 instead of 1981, you're on your own. When your flying fingers come out with a date like June 33rd, you'd better have an understanding lawyer to bail you out of that contract, because the spell checker won't catch it.

Next, spell check programs check spelling, they do not correct it. They will bring the suspect word to your attention, but you must accept it, correct it, or add it to the dictionary in the space provided for your own personal jargon.

Out of context words will not be recognized. If you keep using "their" for "there", or vice versa, you are beyond the help of a mere machine.

## Speed

The success or failure of a spell check program could sometimes be based on its speed of operation. Some of the original spell checkers took as much as three minutes to look over a standard 1200 to 1500 character page. You could live with slow speed for the odd letter, but what happened when you wanted to check a 17 page contract? It could take almost an hour, and that was in no way acceptable. General use of machine language and some extremely sophisticated sort routines have brought the speed of some of the current products down within reason. Fifteen to twenty seconds per standard page is not unusual now, but that, you must remember, is for an error free page. Add the time necessary for you to accept or correct all suspect words. The more errors or suspect words, the more time will be needed to process them.

## WordCheck, A REVIEW

WordCheck, an aid to spelling, is a product of Micro Computer Industries, Ltd. of Fort Collins, Colorado. It has been designed to accept WordPro 3 and 4 files, WordPro 2 Plus (disk version) files, and WordPro 3 Plus and 4 Plus files for processing. The WordCheck package consists of a 2K Eprom, a WordCheck disk and four pages of instructions. The Eprom fits in the $9000 slot and contains a substantial amount of active machine code. The disk contains four program files and three sequential files. The "Words" and "Table" files and the three sequential files "a-g", "h-o" and "p-z" are used in the normal operation of WordCheck. "Update" and "Sort" are used to add and delete words from the dictionary files, and to create new "Table" files. The four pages of instructions may seem light at first, but the program is so simple and so straight forward to use, that the four pages are completely adequate.

## Looking For Errors

First, place your WordPro file disk in drive 1 and the WordCheck disk in drive 0. Then punch in the WordCheck enabling SYS command to start things going. WordCheck will ask for a WordPro file name. This must be an exact, letter perfect file name without resorting to wild cards (?) and jokers (*). Once the file name is input and accepted, WordCheck carries on, automatically chaining through global files looking for suspect words. The program processes each file three times, once for each of the three dictionary files "a-g", "h-o" and "p-z".

## Updating The Dictionary

Next, you may choose to turn your printer on and make a hard copy of the list of suspect words, or just have them appear on the screen. The words that the program has been unable to identify now appear on the screen one at a time, and as you accept them, reject them or add them to the dictionary, are listed by the printer if you have so chosen. Initially, the printed list is fairly important and will assist in tailoring the WordCheck dictionary to your style of vocabulary. When we ran this review through WordCheck, our printed list of unrecognized words contained words like wordprocessor, dictionary, collegiate, boasts, wonder, crunching, megabyte and jargon. These words were all added to the dictionary as being words that we will likely use quite often. WordCheck also pointed out a few spelling errors and typos. Words added in this manner do not immediately get placed in the active word table, but are placed in a temporary file awaiting the running of the "Update" program which will make a new, expanded word table to include the new words.

## Observations

The WordCheck review to this point is a fairly lengthy 139 line (3■ page) WordPro 4 Plus file. The WordCheck program we used was fresh out of the box with the minimum possible dictionary. Here's what happened when we ran our review through WordCheck:

First pass (looking for errors) - 2 minutes

Found 57 suspect words, 2 of which were misspelled or typos, 10 were personal or place names, and 16 were personal jargon that along with the remaining 29 should really be put into the dictionary.

Making hard copy list of suspect words - 3 minutes

Sorting - 2 minutes

Updating the 3 dictionary files - 6 minutes

After we added those 45 words to the dictionary and corrected the two errors, we ran WordCheck again to see if there was a noticable improvement in the elapsed time. We could have put in the personal or place names, but we didn't because you've got to draw the line somewhere.

Here's what happened when we ran the same section of this review through WordCheck with a slightly expanded dictionary:

Second pass (looking for errors) - 1 minutes

Found 11 suspect words, including 5 place names, 5 proper names, and "tortious" which I really didn't think I needed in my dictionary either.

Making hard copy list of suspect words - 1 minutes

Sorting - 2 minutes

Updating the 3 dictionary files - 6 minutes

Note the significant decrease in the time taken to process fewer suspect words. Sorting required the same length of time, and updating took slightly less. Adding words to the dictionary was very simple, even the first time we tried.

## Conclusions

The programmer will recognize WordCheck as a well designed and executed, "state of the art", "hybrid" program, using machine language for speed and Basic for convenience. WordCheck is supplied as a "turnkey" program, without source listings or technical documentation. It is not intended to be changed or modified by the dealer or end-user, except as allowed within the running program.

The end-user will find WordCheck simple to use and functional in concept. Short, single page letters and memos can be checked faster manually, but WordCheck truly hits it's stride when checking long, multi-page documents. We keep a special WordCheck for documents and contracts that has all the legal jargon tucked away in it's dictionary, and another one for checking technical manuals with lots of computer terms in them. A third is used for personal correspondence that recognizes all my own short forms and slang terms.

WordCheck is available from dealers in Canada, United States and in the British Isles, and is now well worth your consideration.